

Technical Report Submission to JALA

Original Submission Date: 12/13/2002

Revised: 01/09/2003

Title: LabRAT (Laboratory Rapid Automation Toolkit): a flexible and robust peer-to-peer architecture with XML based open communication for Laboratory Automation

Authors: Roger L. McIntosh
Amersham Biosciences
928 E. Arques Avenue
Sunnyvale, CA 94085
408-773-4882
roger.mcintosh@amershambiosciences.com

Alfred Yau
VI Technology
c/o Amersham Biosciences
928 E. Arques Avenue
Sunnyvale, CA 94085
408-773-1222
alfred.yau@amershambiosciences.com

Funding: NHGRI grant #1 R24 HG02211-01

ABSTRACT

Traditional lab automation systems are highly centralized: dispatch and coordination of activities are mediated by a system controller, usually via a single, monolithic control procedure. This approach, while conceptually simple, makes changes to the system difficult; adding or removing instruments and functionality can be a daunting task. In addition, most automated systems are tied to particular development languages and protocols, making operation in heterogeneous environments (i.e. the real world) problematic, since instrument software comes in many different implementations.

We present a peer-to-peer architecture for lab automation, using an XML based communication protocol. The architecture consists of peer instrument servers, an XML communication layer, and an open control center. Each instrument peer can control, be controlled by, and communicate information to other instrument peers to fulfill the automation task. Our protocol is based on XML-RPC, a lightweight communication standard built atop HTTP. This provides an open and flexible means of peer-to-peer interfacing. The control center serves as a convenient, web-based interface to manage the instruments. The automated procedure can be distributed across all available instrument peers (each instrument assigned a set of responsibilities); the controller implements a limited set of high level instructions. The software components included in our prototype system are implemented in various programming languages, including Java, C/C++, Visual Basic, and LabVIEW. Our approach facilitates rapid development of laboratory automation systems.

Keywords: automation, communications protocol, peer-to-peer, command and control, distributed computing, web services

INTRODUCTION

Building an automated system for laboratory use poses an inevitable design dilemma: should one choose a primary automation supplier and “lock in” to the supplier’s proprietary technology platform or is it preferable to choose individual components and integrate them via custom software? Each choice offers a range of sub choices and a set of consequences, potentially both positive and negative. The primary supplier may offer custom plugins or drivers for a range of popular hardware and may offer custom services to create such enhancements if not already available; alternatively, many suppliers offer developer tools to allow third parties to write code to their proprietary format. If an independent path is chosen, the integrator may operate at any of several different levels of granularity, from integrating individual low level devices to buying major subsystems and “gluing” them together via high level code (and perhaps custom mechanicals and electronics). Whichever path is chosen, the automation designer faces major tradeoffs – if the primary supplier route is chosen, additional elements may be difficult or expensive to add later; if instead, lower level components are integrated independently, much more effort (and perhaps cost) will be incurred as the cost of flexibility. A primary supplier usually offers comprehensive system support packages; with a low level approach, the integrator is in effect developing his own proprietary technology to meet his precise needs, but will likely incur full support responsibilities for the resulting system. Such was the dilemma our research group recently faced when confronted with the task of automating a set of instruments to perform sub-microliter sample preparation for genomic sequencing. We decided to pursue a third, less obvious approach.

Our approach was to design our own high level command and control protocols and software, but to do so in a manner that facilitates easy and convenient integration of existing instruments by supplying a simple “front end” or “wrapper” to the instruments. By choosing our technologies carefully and borrowing heavily from open standards and code, we were able to define a set of protocols and base code that now permits very rapid integration of new components and methodologies into our automation system. In addition, the highly modular nature of our architecture, along with standardized and clearly defined interfaces, reduces the cost and effort required to troubleshoot and support automated systems built with our framework. We describe here the techniques we used and the approach to various elements of the architecture. Since much of the underlying code is free and readily available and since we are placing our high level protocols into the public domain, the same approach should be viable for many other labs. We believe that our protocols are much more approachable and tractable than existing lab automation protocols known to us, such as Laboratory Equipment Control Interface Specification^{1,2} (LECIS). In addition, we assert that our approach offers several distinct advantages over LECIS and most of the proprietary automation interfaces available, particularly in its support for distributed automation logic and the ability to dynamically scale the system simply by adding peer-to-peer^{3,4,5} instrument nodes.

Peer-to-Peer networking is a distributed computing concept with a long history that has recently found new favor in file sharing and other Internet based applications. The term refers to both the topology and the architecture of computers and software within a

networked system. In a traditional client/server environment, roles are well defined: clients make requests for specific services on a predefined entity (usually external to the client) and servers are dedicated to fulfill the role of servicing such requests. Usually, each computer or node within the system acts as either a client or as a server. In contrast, each node within a peer-to-peer network is designed to both service requests of its peers and to make requests on its peers. The computers within such a system all hold the same “status” within the network hierarchy – they are all peers. As we show later in this report, the peer-to-peer approach provides unique advantages in terms of flexibility and scalability to the automated system designer.

BACKGROUND AND GOALS

Amersham Biosciences produces a wide range of instrumentation and reagents for the biotech industry. Our lab, a research group within the company, is tasked with design of new techniques and prototype instrumentation. One such set of instrumentation and associated lab protocols involves sub-microliter sample preparation for genetic sequencing; the steps include thermal cycling, reagent addition, and centrifugation. A prototype set of manually operated (but computer controlled) instruments were produced as an early phase of the project. A second project phase required automating all elements of the system, including both the custom instruments and several commercial instruments, such as an automated centrifuge and a refrigerated microtiter plate carousel. We squarely faced the automation dilemma outlined earlier: buy and lock in or build and lock out. Since the company is often faced with the same decision, we chose to build a general framework for the immediate project upon which a wide range of future projects could also be built. The goals of the automation framework project were:

- Provide a flexible framework for automation of lab protocols for our immediate project (sample prep for genomic sequencing). The framework should include all software and hardware elements needed to accomplish the targeted lab protocols.
- Build a software framework that would include “wrapper code” for existing instrument control software to allow rapid integration of instruments. The project included instruments with existing control software written in C++, LabView, and Visual Basic; the goal was to interface to this existing software with minimal effort.
- Provide the basis for future automation projects involving instruments of nearly any type and with control software of nearly any description. The key to accomplishing this goal is to ensure a high level of modularity of both the software and hardware and to standardize the instrument interfaces while allowing for a broad range of instrument types.
- Define a general instrument command and control protocol to facilitate interoperation of instruments over a standard Transmission Control Protocol over Internet Protocol⁶ (TCP/IP) network. Since TCP/IP is now the lingua franca of network communications, this is the obvious choice. For instruments employing other interfaces (RS232, RS485, CAN), a low cost PC or microcontroller provides the bridge to the network.
- Design the system as a peer-to-peer instrument network to allow for either distributed logic or operation from a central controller. An instrument should be able to request services of another instrument without involving the central controller. The human operator should be able to call upon high level instrument functions when designing a

protocol without having to deal with low level issues like individual robot move routines that should be implied in the method call.

- Build a flexible central process controller for high level protocol definition. Although the initial control center implementation is required to support only simple sequential operations, allow for enhancement to a full dynamic scheduling system.
- Design the modules to communicate via Hypertext Transfer Protocol⁷ (HTTP). This includes operation of the control center to allow protocol definition, status checking, starting of analyses, etc. using nothing more sophisticated than a standard web browser (e.g. Microsoft Internet Explorer).

In addition to these “framework” goals, we required several project specific modules and routines, most of which (like the robot module and its associated move subroutines) have general applicability to a wide range of lab automation projects.

SYSTEM DESIGN OVERVIEW

Based upon the design goals stated above, we first surveyed the domain landscape to determine what was currently available and which existing techniques and technologies might best serve our needs. The most conspicuous candidate on which to base our lab automation protocol was LECIS. In many ways LECIS appears at first blush ideally suited as a control standard to serve our stated needs. LECIS defines a standardized set of equipment behaviors and interactions covering a broad range of command and control functions. The LECIS standard defines comprehensive state models as well as standard messages and responses. As we looked closer, however, a few key difficulties with strict LECIS conformance arose:

1. The American Society for Testing and Materials (ASTM) standard definition is fairly abstract and is intended to cover a very broad range of functionality. For strict conformance, a set of required interactions and states must be supported, whether they are actually relevant to the instrument or not. For many instruments, this imposes unnecessary overhead and requires that “dummy” functionality be coded solely to satisfy the standard.
2. No reference LECIS implementation exists and the ASTM standard is neutral with respect to the actual communication mechanism (hardware and associated software to accomplish messaging between instruments). While this makes LECIS flexible and allows the developer freedom to use nearly any communication technology, we wish to define all elements of our protocol needed to implement a working system and to assure compatibility of modules conforming to the protocol.
3. LECIS defines a hierarchical control model and offers distributed control based upon the defined Task Sequence Controller (TSC) hierarchy. Since one of our goals was to include provision for a variable degree of distributed control, thereby removing linked dependencies from the human operator level, we prefer the flexibility of a peer-to-peer control model.
4. Since there appears to be a very low level of market acceptance of LECIS, no advantage is currently gained by conforming to this abstract standard (i.e. one cannot easily find components to “plug in” to a LECIS compliant system).

The second issue (reference implementation and connection standards) is currently being addressed by the Object Management Group⁸ (OMG) effort to revamp the LECIS

standard, but this effort is focusing on a Common Object Request Broker Architecture⁹ (CORBA) reference implementation, which for our purposes is too complex and of limited practical value. We believe that the future of distributed computing clearly points toward peer-to-peer nodes communicating via web services over HTTP; CORBA no longer represents a viable solution in this arena.

We decided to borrow the core LECIS concepts (the instrument state model and the key message/response definitions), but to construct our own, simpler and more practical instrument control specification (subsequently named LabRAT, for Laboratory Rapid Automation Toolkit). Our model is largely a subset of the ASTM standard, but also includes clear definition of the physical and communication layer, which LECIS does not address. We chose a web services approach based upon Extensible Markup Language – Remote Procedure Calls¹⁰ (XML-RPC), a simple, open, stable, and widely adopted standard for exchange of Extensible Markup Language¹¹ (XML) based messages over HTTP, as our message transport mechanism. Many developers familiar with HTTP based messaging will immediately ask why XML-RPC was selected in preference to Simple Object Access Protocol¹² (SOAP). The answer to this is reminiscent of the answer given regarding use of LECIS: although SOAP includes the word “simple” in it’s name, XML-RPC is simpler and eliminates the unnecessary (for our application) functionality and overhead inherent with SOAP. XML-RPC is a subset of SOAP and has the significant advantage of being stable and “frozen” (SOAP is still rapidly evolving), while offering a straightforward transition path should we wish to support SOAP enabled peers in the future. We believe that the web services communication model we have adopted, with XML-RPC as the core messaging protocol, is far easier to understand and work with than is CORBA and is far more universal than many language or platform specific protocols such as Remote Method Invocation¹³ (RMI) or Distributed Component Object Model¹⁴ (DCOM).

Our current control center, which defines the laboratory protocol to be executed and makes the necessary instrument method calls to implement the protocol (along with error handling, event logging, primary system GUI, etc.), is coded as a web based application. This provides many benefits over a more traditional, standalone executable application:

- The application design fits naturally with the system architecture as simply another node within the peer-to-peer network running the LabRAT protocol.
- The operator can access the controller using a standard web browser from any workstation on the local network or even across the Internet (if so configured – security concerns may argue against this arrangement). No special training is needed concerning operation of the GUI since browser operation is familiar to almost everyone.
- The system can be configured to send alerts and error messages via email to the human operator; the system can readily employ a variety of Internet based communication protocols for sending notifications to external systems.

It is important to note that our approach heavily leverages open source protocols and code developed over the past few years to support web services and applications. The tremendous quantity and quality of work done by the communities of open source web

and peer-to-peer developers have allowed us to quickly construct a sound scaffold upon which to build our lab automation architecture. An added benefit of this approach is that we achieve a high degree of automatic compatibility with the technologies currently being deployed to support a broad range of web based services. This is highly beneficial both for ease of system maintenance (the pool of developers familiar with our design and programming practices is very large) and for future extensibility (we will continue to take advantage of the rapid advances being made by the open source community).

SYSTEM ARCHITECTURE

The LabRAT framework is comprised of three primary elements: a communication layer, a peer server for each instrument in the automated system, and a control center. These three elements represent a general framework upon which specific automation systems can be rapidly built.

The communication layer is based upon XML-RPC and implements code to flexibly and reliably transfer messages between the system nodes. It encodes the LabRAT command and control protocol, which was inspired by the LECIS protocol.

The instrument peer server provides a bridge between the local instrument control software and the rest of the LabRAT system. This takes the form of “wrapper” or inheritable code, allowing the local instrument software to plug into the LabRAT system. In addition, each instrument peer server has associated definition files that describe the dependent tasks (if any) required for the instrument to fulfill each method request. Typical dependent tasks include robot calls to deliver a plate or other resource before the local instrument method runs and another robot call to retrieve the resource upon completion of the local method.

The control center is an optional element of the LabRAT system that provides a range of services for the instrument peer servers and enables the operator to interact with the system dynamically. Using the control center’s web based interface, the operator can design and compose each automation process based upon the available instrument peer servers and their locally defined methods (we use the term “process” to refer to a complete automated laboratory protocol). The control center executes processes by dispatching requests to each instrument peer server as needed. In order to communicate with each other, the instrument peer servers rely upon the control center’s services for locating available instrument peer servers. The control center is optional because the automation designer or integrator may choose to follow a purely distributed model, in which case the functionality of the control center is reduced to initiating a single request on a single instrument server, which can be accomplished via the local instrument interface itself.

Figure 1 illustrates the relationships between the system elements discussed.

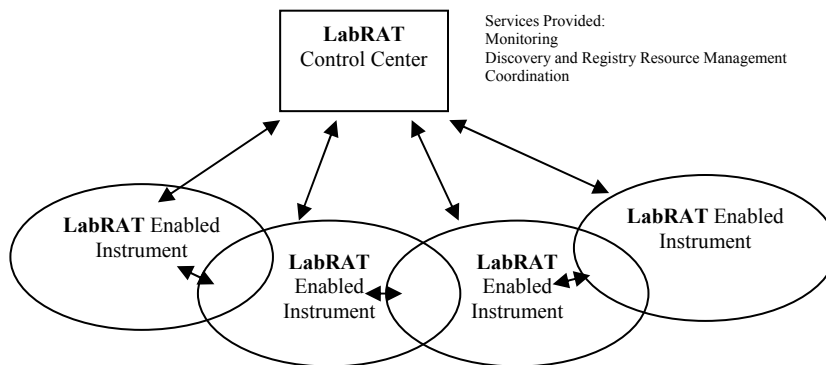


Figure 1: Overall architecture of a LabRAT automation system

The three LabRAT elements work interdependently to complete each defined process. To execute a process in a LabRAT enabled system, the operator first defines the process, then submits it to the control center's execution engine. The control center dispatches the appropriate requests to each instrument peer server in order to fulfill the request. Upon receiving a request, the instrument peer server initiates and requests services of other instrument peer servers as needed to fulfill its dependent tasks (specified within the local definition files). For example, suppose an operator wishes to execute the following simple protocol, part of an actual sample prep routine.

1. Aspirate raw material from a stored microtiter plate into a capillary cassette.
2. Bind and denature DNA via thermocycling.
3. Wash to remove extraneous material (leaving bound DNA behind).
4. Aspirate sequencing cocktail from a microtiter plate.
5. Execute terminator reaction via thermocycling.
6. Transfer product to a fresh microtiter plate by centrifugation.
7. Transfer final product microtiter plate to refrigerated storage.

The operator translates the above protocol into a LabRAT process as listed below.

- Step 1: Specify Storage method to perform sample aspiration.
- Step 2: Specify Thermocycler method to perform binding/denaturation.
- Step 3: Specify Washer method to perform ethanol wash.
- Step 4: Specify Storage method to perform cocktail aspiration.
- Step 5: Specify Thermocycler method to perform terminator reaction.
- Step 6: Specify Centrifuge method to perform capillary-to-plate transfer.
- Step 7: Specify Storage method to perform plate storage.

Note that, although this is an automated process, *no calls to a robot have been made*. Operation of the robot is implicit in each instrument method.

The list of steps is saved under a process name, along with appropriate parameters defining such details as plate storage locations and instrument specific settings. Defining a LabRAT process is simply a translation of the manual protocol into a sequence of instrument method calls. All dependent tasks such as moving a sample from one instrument to another by the robot are specified as part of each instrument server's definition files rather than being explicitly coded in the process itself; thus many of the difficult details about the automated process are defined once, associated with the local instrument method, and need not be considered subsequently when setting up the high level processes. As a result, a LabRAT process is very easy for the operator to understand and maintain; it is unnecessary to list detailed, subordinate tasks such as movement of items by the robot, at the process level. This represents a tremendous advantage over traditional automation approaches.

COMMUNICATION PROTOCOL

The communication layer enables message exchange between the control center and the instrument peer servers, as well as among instrument peer servers themselves. The LabRAT protocol is based on the XML-RPC protocol, which in turn relies upon HTTP, as shown in Figure 2.

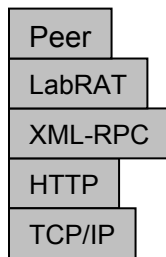


Figure 2: LabRAT protocol stack

XML-RPC is a simple standard for data exchange using XML over HTTP. Since it is based upon HTTP, every request generates a response. XML-RPC requests and responses are carried as specialized HTTP content. Instead of transmitting HTML content, we transmit XML content inside HTTP packets. XML-RPC is easy to learn and use; every payload transmitted is readable by human and standard XML parsers. The remote procedure call (RPC) semantics are ideal for request and response based applications such as remote instrument control. Many excellent open source implementations of XML-RPC are available in all popular programming languages. A simple XML-RPC request and response pair is illustrated in Figure 3.

```
<?xml version="1.0"?>
  <methodCall>
    <methodName>examples.getStateName</methodName>
    <params>
      <param>
        <value><i4>41</i4></value>
      </param>
    </params>
  </methodCall>
```

```

    </params>
  </methodCall>

```

Figure 3a: Example of XML-RPC request

```

<?xml version="1.0"?>
  <methodResponse>
    <params>
      <param>
        <value><string>South Dakota</string></value>
      </param>
    </params>
  </methodResponse>

```

Figure 3b: Example of XML-RPC response

Communication Format

The LabRAT protocol defines standard request and response formats to be transmitted via XML-RPC. Each request and response must include a message identifier and message state. This permits an arbitrarily long message chain “conversation” to be conducted between two entities. The message state indicates the ordinality of a particular message within the message exchange: Init, Continue, or Final. The two classes of messages, Command and Response, map directly to XML-RPC’s request and response respectively. Command indicates a requested operation. Response returns a reply or functional consequence of the requested command. Protocol errors are handled by the lower level XML-RPC fault code mechanism. High-level errors are represented in the response, allowing the calling module an opportunity to take remedial action.

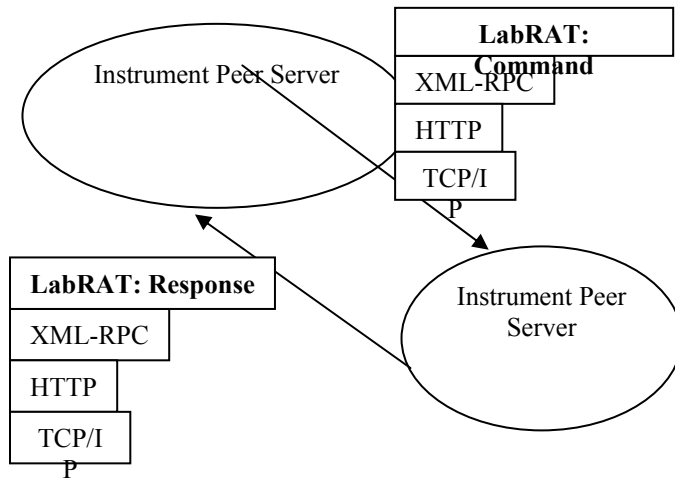


Figure 4: LabRAT communication flow

Command Format

A command message consists of the following parts:

<CommandName> takes the form *CommandName [string]: {instrumentType}.command*

Example: `cycler.runMethod, robot.eStop`

<**CommandParameter(s)**> takes the form *CommandParameter(s) [struct | primitive type]*
 This represents a list of parameters for the command.

<**MsgID**> is a string that encodes a unique value for each message in the chain.
 It is a 16-byte string in Hex format: `xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx`
 Example: `7ac0f385-81a7-4b4d-a43b-adac0bd1a2f4`

<**MsgType**> is a string representing the message ordinality: `init, continue, or final`.

<**CallerURL**> is a string that encodes the caller's Uniform Resource Locator (URL)
 Examples: `http://127.0.0.1:8899` or `http://123.23.23.01:123/cgi-bin/rpc2.vi`

<**CallerInvokeName**> is a string that encodes the caller's invocable name
 Example: `controller`

Table 1 lists the primary LabRAT commands. Adding commands is permitted.

Command Name	Description	Parameters	Response
RunMethod	Run a method defined in the instrument	MethodInfo:struct { MethodName: string Method param1: string Method param2: string ... Method paramN:string SampleID: string Mode: string }	Ack/Nack
ReadState	Request for instrument status	None	Status/Nack
Estop	Stop the running method	None	Ack/Nack
ReadMethod	Request for a method file	MethodName:string	File/Nack
RunTime	Request for run time on a method	MethodName:string	Time/Nack
Update	Update expected runtime	ExpectTime: int, in second	Ack/Nack
Done	Indicate completion of subsequent task	None	Ack/Nack
Synch	Indicates that instrument is paused and waiting for a return synch with the same SynchNumber, or a return message in response to such an originating message.	SynchNumber: long, 1 -> N	Ack/Nack
Alert	Error/Warning/Emergency	AlertInfo:struct { Type: string [Error Warning Emergency] Code: string Source: string Msg: string }	Ack
Echo	Echo	None	Ack
Join	Join the automation instrument cluster	None	Ack/Nack
Leave	Leave the automation instrument cluster	None	Ack/Nack
Query	Query for a specific instrument's address	Name: string	Location/Nack

Response Format

A response message consists of the following parts:

<ResponseType> is a string encoding one of the replies shown in Table 2.

<ResponseContent> takes the form *struct* | *primitive type* and represents response data

<MsgID> is a string that identifies the message chain (same as command value)

<MsgType> is a string representing the message ordinality: init, continue, or final.

Table 2 lists primary LabRAT responses. Adding responses is permitted.

Response Type	Description	Response Content
Ack	Indicates correct operation	ExpectTime: long (-1 means no known or meaningful expect time), in seconds
Nack	Indicates error when trying to initiate the operation	ErrorMsg: string
Time	Run time information	Time: int, in second
File	Indicates File content in the response	FileContent: binary
Status	Indicates Status information in the response	StatusContent: struct { Mode:Local Remote State: Idle PreMethod Method PostMethod SampleID:string RemainingTime: int, in second ExpectTime: int, in second }
Location	Location url for the queried instrument	url: string

Protocol Performance

To answer questions regarding the LabRAT protocol's performance and its applicability to automated instruments with time critical messaging requirements, we tested the effect of heavily stressing multiple instrument nodes with rapid-fire requests. We found that the performance of LabRAT's XML-RPC based approach is quite adequate for most lab automation applications. Performance was measured by timing communication exchanges between two computers within a local LabRAT instrument network. We measured the time delay between a RunMethod message sent from the first computer and the corresponding response received from the second. To test scalability, we configured our sending peer (a 600MHz PII system) to generate multiple concurrent message requests. To characterize the effect of computer hardware speed on communications efficiency, we compared the performance of two different receiving peer computers, one running at 500MHz and the other at 900 MHz. As shown in figure 5, processing time scales linearly with the number of concurrent peer servers. In typical LabRAT automation systems, each peer instrument deals with only a few peer servers simultaneously. In such cases, processing time is usually in the ten-millisecond range.

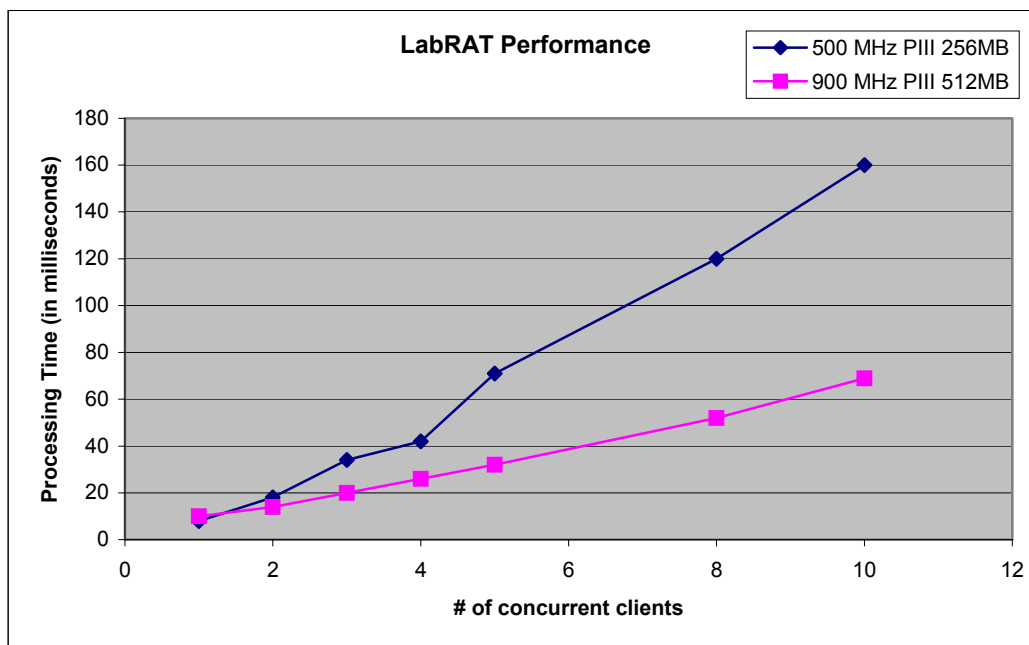


Figure 5: LabRAT communication performance

INSTRUMENT PEER SERVER

Each peer server provides an external interface or “wrapper” for the control software of a particular instrument – the server receives and processes XML-RPC requests and issues requests to other servers. Each peer server’s behavior is governed by a set of definition files and a standard state model. The LabRAT protocol specifies three types of definition files: pre-method, post-method, and remote-method. A pre-method file contains all of the pre-operation tasks (i.e. tasks to be carried out before the local instrument method runs) and specifies their sequence of execution. Similarly, a post-method file defines the post-operation tasks. Finally, a remote-method serves as a binding mechanism between various pre-method and post-method tasks. To illustrate this concept, consider step 2 in the simplified protocol presented earlier. Step 2 requests a thermocycler to execute a specific temperature cycling method. In order to carry out the method, one pre-operation task and one post-operation task must be completed before and after the local thermocycler method runs. In this case, the pre-operation task ensures that a capillary cassette containing sample material is delivered to the thermocycler. The post-operation task removes the cassette from the thermocycler when the local method is completed. These tasks are defined in pre-method and post-method files; the remote-method file serves to list the pre-method(s) and post-method(s) required for this method (it is essentially a “metafile” pointing to the other files).

Each request is executed according to a predefined state model. The model defines three major states: premethod, method, and postmethod. In the pre-method state, the server executes operations defined in the premethod file(s). In the method state, the server invokes the instrument control software’s local instrument method. The post-method state invokes all cleanup operations defined in postmethod file(s).

Since instrument behavior is based on machine state and defined in associated files, integration with control software is very straightforward. LabRAT implements the state model within a set of base objects. By extending these base objects, the developer can rapidly integrate instrument control software written in a variety of software languages.

Although specific details of the implementation are outside the scope of this report, it should be noted that the LabRAT code base employs a standard object oriented approach and common design patterns to provide a flexible framework for integrating control software into an automation system.

The LabRAT design distributes an arbitrarily complex sequence of operations into small, manageable modules. Most importantly, each instrument server maintains its own set of methods and definition files. The centralized process controller stores no low-level details of the instruments it manages nor does it need any knowledge of specific dependent operations required to perform instrument tasks. This self-managing and semi-autonomous behavior provides tremendous flexibility for coordinating large and complex laboratory protocols and conceptually simplifies the human operator's job of defining and maintaining such protocols.

CONTROL CENTER

The final element of a LabRAT powered system is the control center. The control center provides an interface between the system and the human operator; it also provides essential, centralized services for peer instrument servers. The center has a web-based interface as shown in Figure 6, which allows the operator access via a standard web browser. The interface exposes several functions, including process management, peer server management, execution management, etc. The operator can use the interface to design and define high level processes, monitor the status of any peer server in the system, and submit and monitor process executions.

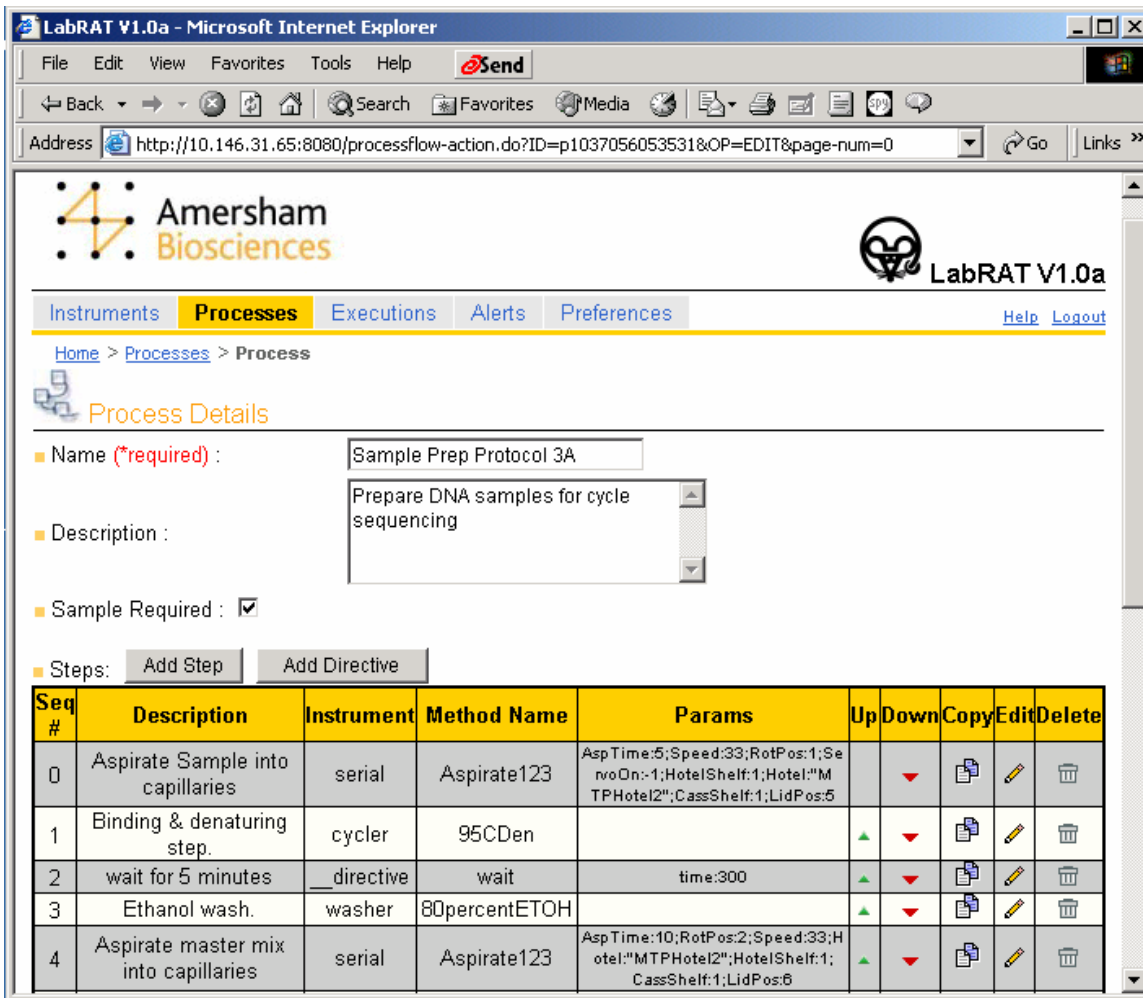


Figure 6: Control center web user interface (showing process definition)

In addition to providing the primary user interface for the system, the control center offers a set of services for each instrument peer server. Each peer server discovers the presence of its peers through the control center's registry function. A peer server sends a "join" command to the control center when it is available to offer its services and sends a "leave" command when it can no longer provide services to the system. The control center continuously tracks each peer server's location and availability and provides this information to peers in need of services. It also sends "heartbeat requests" to all known peer servers to ensure that information regarding each peer server's status is up-to-date and accurate (should a peer server die unexpectedly, this is the mechanism by which the control center is informed).

Another function of the control center is system error handling and logging. Every received warning or error is logged and optionally forwarded to a specified email account. An appropriate error recovery routine is executed based upon the particular error received. Such error handling routines can be customized by the system designer and can encode a wide range of recovery or reaction strategies.

Coordination of multiple process executions is another important control center task. When the operator submits a process, the control center places it into a queue and executes it according to availability and scheduling. The operator can monitor each execution either locally or remotely (over the network or Internet). Ongoing process execution can be stopped by the operator or cancelled before execution begins.

Finally, the history of every process execution and error event is stored to an embedded database. The operator can easily review the history of all system activity; this is an enormous aid in designing and troubleshooting automated processes. Although the LabRAT control center is similar in some respects to a conventional centralized automation server, it defines and governs only high-level system operations. The control center represents the “glue” that allows multiple, independent peer servers within the system to cooperate in carrying out a high level set of process instructions.

FUTURE DIRECTION

Our current control center implements only sequential task execution, although parallel operations are possible by coding the logic in a distributed manner. We plan to add a task scheduler to the control center to allow for coordinated, parallel task execution, along with dynamic rescheduling based upon realtime events.

Other anticipated extensions include:

- The capability to exchange instrument method file libraries between remote instruments over the LabRAT protocol (useful to synchronize method file libraries across installations at multiple sites).
- Development of a general “intelligent agent” module to allow for greater front end automation intelligence for the instrument modules. This may include code to allow for meaningful negotiations between instruments in order to determine how best to carry out a requested action without intervention by the control center or the human operator.
- Support for additional message exchange protocols. While XML-RPC represents an ideal blend of simplicity and functionality, systems already designed around the SOAP protocol may benefit from the addition of SOAP support in LabRAT.

CONCLUSION

We have described our peer-to-peer, XML-RPC based rapid automation framework for laboratory instrumentation known as LabRAT. This framework leverages much of the excellent work recently made available by the open source community to support web services and peer-to-peer networks. Our approach represents a middle ground between adopting a turnkey set of proprietary technologies from a single automation supplier and developing a custom set of components for each specific automation application.

LabRAT is a reusable framework enabling rapid incorporation of new instruments or automation subsystems based upon a broad range of control software into a system, thereby allowing definition and execution of a wide variety of laboratory protocols. Compatibility with mainstream practices, along with a clear path for extensibility is assured, since it is built upon widely adopted open source protocols and code.

We believe that our protocol represents an excellent balance between simplicity and ease of use on the one hand and flexibility and adaptability on the other. The command and control protocols, instrument peer server wrappers, and control center are designed to be as simple as possible and amenable to rapid reconfiguration and enhancement. LabRAT's peer-to-peer model enables the flexibility to locate control logic either in a central location, as is traditionally the case with lab automation systems, or to distribute the logic among the instrument peer nodes. The architecture supports either sequential or synchronized parallel operation of tasks. Although our current control center implements only sequential task execution, the addition of a task scheduler to allow for parallel, coordinated task execution and dynamic task rescheduling represents a straightforward enhancement.

ACKNOWLEDGMENTS

We wish to thank the NHGRI for funding portions of this work (NHGRI grant #1 R24 HG02211-01). We sincerely thank our colleagues for chemistry support (Corey Garrigues, Daniel Yung), mechanical engineering support (Dave Roach, Tom Yang), electrical engineering support (Bob Loder), and managerial support (Stevan Jovanovich, Sharron Penn).

REFERENCES

1. "Standard Specification for Laboratory Equipment Control Interface (LECIS)", ASTM E1989-98
2. "LECIS Home Page", <http://www.lecis.org>
3. Peer-to-Peer Working Group, <http://www.peer-to-peerwg.org>
4. Peer-to-Peer Forum, <http://openp2p.com>
5. "An Overview of Peer-to-Peer", Rollins, S., CS276 Guest Lecture UCSB, <http://www.cs.ucsb.edu/~srollins/talks/p2ptutorial.ppt>
6. "A TCP/IP Tutorial", January 1991, <http://www.faqs.org/rfcs/rfc1180.html>
7. "Hypertext Transfer Protocol Specification", <http://www.w3.org/Protocols/HTTP/HTTP2.html>
8. OMG LECIS, http://www.omg.org/techprocess/meetings/schedule/LECIS_RFP.html
9. "CORBA BASICS", <http://www.omg.org/gettingstarted/corbafaq.htm>
10. "Simple Object Access Protocol (SOAP) 1.1 Specification", May 8, 2000, <http://www.w3.org/TR/SOAP>
11. "Introduction to XML", http://www.w3schools.com/xml/xml_what.asp
12. "XML-RPC Specification", Winner, D., June 15, 1999, <http://www.xmlrpc.org/spec>
13. "Remote Method Innovation Home Page", <http://java.sun.com/products/jdk/rmi/>
14. "Distributed Component Object Model Home Page", <http://www.microsoft.com/com/tech/DCOM.asp>